64 PLUS 4



& AMIGA

LISTOPAD 1991 ISSN 0867-3918 INDEKS 377112 CENA 8.000 zł

MIESIĘCZNIK UŻYTKOWNIKÓW KOMPUTERÓW COMMODORE



OD REDAKCJI

Wadliwa dystrybucja „64 plus 4 & Amiga" przez przedsiębiorstwo RUCH jest przyczyną kłopotów, jakie mają czytelnicy chcący nabyć nasze pismo. Zdarza się, że otrzymujemy jako zwroty NIETKNIĘTE paczki zbiorcze. Tymczasem czytelnicy sygnalizują, że do wielu kiosków nie dociera ono wcale. Dlatego:

zapraszamy wszystkich chętnych do prowadzenia kolportażu „64 plus 4 & Amiga" (kluby, studia i sklepy komputerowe, księgarnie, osoby indywidualne itd.) do współpracy!

Oferujemy korzystne warunki!

Zainteresowanych prosimy o kontakt z działem dystrybucji pod adresem: Przedsiębiorstwo ABUK, 87-200 Wąbrzeźno, ul. 1 Maja 33.

NOWOŚĆ!

PUBLIC DOMAIN PACK dla COMMODORE 64 NA KASECIE!

Jeszcze w tym roku ukażą się cztery kasety!

Na naszych zestawach znajdują się programy, które do tej pory ukazały się na dyskach PDP, oraz wiele nowości!

Pojedyńcza kaseta kosztuje 30.000zł, wykupienie prenumeraty do końca roku (4 kasety) kosztuje tylko 110tys zł.

Wśród wszystkich, którzy zamówią komplet kaset (blankiet wpłaty zamieszczamy na str. 15) rozlosujemy 10 bezpłatnych prenumerat naszego pisma na rok 1992!

Na blankiecie prosimy dopisać PDP-taśma. Spis zawartości kaset wewnątrz numeru.

Przedsiębiorstwo ABUK S-ka z o.o. oferuje państwu **szybką i tanią obsługę reklamową.** Ogłoszenia drobne od osób indywidualnych (do 10 słów) przyjmujemy bezpłatnie. Większe - 1000 zł za słowo. Reklamy ramkowe (minimalny format - 20 cm^2): 1cm^2 ogłoszenia **-8000zł**, cała strona - **3,0 mln zł**; kolor - odpowiednio 100% drożej.

Ogłoszenia przyjmujemy za pośrednictwem poczty (nasz adres - patrz stopka redakcyjna). Treść ogłoszenia z określeniem formatu reklamy (ewentualnie zamówieniem koloru) prosimy nadsyłać listem poleconym wraz z odcinkiem wpłaty. Wpłat prosimy dokonywać za pomocą przekazu pieniężnego na konto Przedsiębiorstwa ABUK, Bank Polska Kasa Opieki SA Oddział w Bydgoszczy, konto nr : 5.09011-400522.7-136-11-111.0. Dołączenie do zamówienia odcinka wpłaty przyspieszy zamieszczenie reklamy.

64 PLUS 4

miesięcznik nr 11(13) listopad 1991
cena 1 egz.: 8000 zł



Wydawca:
ABUK Spółka z o.o.

Redakcja nie ponosi odpowiedzialności za treść ogłoszeń.

Adres redakcji: Redakcja „64 plus 4" 85-166 Bydgoszcz 43 skrytka pocztowa 64

Redagują: Marcin Dudar, Sambor Kuźma, Paweł Sołtysiński, Waldemar Szczygieł (red. nacz.), Krzysztof Kobuz.

Okładka: Piotr Bartz.
Skład: ABUK
Druk: Z.P. Wąbrzeźno

W numerze :

W następnym numerze:

- ARP Library cz. 2
- AMIGA - powrót do BASIC'a?
- Czy C-128D=128D?

Z daleka i z bliska

*** W CO POGRAMY ***

Koła w ruch! Para buch! Jak ta przysłowiowa lokomotywa rozkręcił nam się rynek programów. Gry pojawiają się tak szybko, że niektóre nie zdążą się jeszcze znudzić gdy ich miejsce zajmują już następne. Firmy prześcigają się w kolorowych reklamach i wymyślnych formach promocji. Ale przejdźmy do konkretów. Oto co nas czeka w najbliższej przyszłości.

***

- Dobrze znana Sierra Online przygotowuje się do wielkiej promocji trzeciej części serialu „Police Quest". Tym razem fabuła jest otoczona tajemnicą i z reklam nie można wywnioskować co będzie naszym celem w tej grze. Jednak z pewnością ciężko będzie się od niej oderwać, gdyż w odróżnieniu od poprzednich części tu mamy do czynienia z grafiką na pograniczu digitalizy i rysunku. Warto będzie kupić oryginał, by po kilku dniach grania nie mieć przykrych przygód z błędnie skopiowanymi dyskami. Oprócz kontynuacji:Leisure Suit Larry 5 (tak, 5 nie 4!), King Quest 5, Space Quest 4; będziemy mogli zobaczyć trzy nowe tytuły: Mixed up Fairy Tales, The castle of Dr.Brain, Eco Quest. Na uwagę zasługuje fakt, że firma postanowiła wznowić stare tytuły w nowej szacie graficznej. Jak narazie gotowe są pierwsze części gier „Leisure Suit Larry" i „Space Quest". Nic tylko cierpliwie czekać lub wybrać się w kierunku najbliższego normalnego sklepu z programami (czytaj: na zachodzie).

***

- Firma Domark znana z ostatnio z wielu konwersji gier salonowych przygotowała dla nas dwa tytuły. Pierwszy z nich to „Mig-29M Superfulcrum", czyli unowocześniona wersja ich poprzedniego symulatora lotu. Ci wszyscy, którym podobała się pierwsza część mogą w ciemno kupić drugą i napewno nie będą tego żałować. Druga z gier to przygotowany wraz z firmą Tengen konwersja bardzo popularnej gry coin-op pod tytułem „Pit-Fighter". To z pewnością będzie hit i każdy będzie chciał zagrać w tą grę chociaż raz. Dzięki digitalizowanym postaciom walczących gra staje się prawie „symulatorem" walk wschodu.

***

- Uwaga wszyscy entuzjaści Robocop'a. Ocean z myślą o was wydaje trzecią część sagi o metalowym strażniku prawa. Tym razem cały program oparty jest na wypełnionej grafice wektorowej. Oprócz walki z „niegrzecznymi" robotami-ninja, będzie można polatać jetpack'iem oraz prowadzić samochód. Cała gra reklamowana jest jako nowy rozdział w grafice 3D. Zapowiadane są również adaptacja filmu „Hudson Hawks" pod tym samym tytułem, oraz symulator wolnej amerykanki „WWF" z Hulkiem Hoganem w roli głównej.

***

- Electronic Arts zapowiada kontynuacje swoich dwóch najpopularniejszych gier. Pierwsza z nich to „Populos II", w którym poszerzono ilość opcji służących do nękania przeciwnika, oraz ulepszono grafikę. Druga to dysk z danymi do gry „Powermonger" pod tytułem „The World War I". Jeśli ktoś spodziewa się latających samolotów, to wcale się nie myli. Wszystko to ukaże się pod koniec roku 1991.

***

- Psygnosis w grudniu ma zamiar wywiązać się z obietnicy danej fanom gry „Shadow of the Beast". Trzecia jej część (podobno ostatnia) jak zwykle oszołomi nas przepiękną grafiką i całą plejadą przeciwników.

Robert 'Mr.Raf' Turliński

Uczymy się programować (cz.2)

Wskazówka nr 6:
tylko ten, kto wie jak funkcjonuje jego komputer może w pełni wykorzystać jego możliwości.

Dzisiaj przedstawiamy informacje, jak komputer przetwarza zmienne zmienno-przecinkowe i całkowite.

W pierwszej części, szkic rozmieszczenia pamięci pozwolił początkującym zapoznać się z pamięcią użytkownika C-16 i Plus/4. Pamięć tą dzielą ruchome granice, których aktualne adresy można każdorazowo odczytać ze strony zerowej (pierwsze 256 bajtów pamięci).

Pod adresem 45 i 46 zawarte są młodsze (L) i starsze (H) bajty adresu początku zmiennych (patrz rys. 2 w poprzednim odcinku). Adres ten oznacza równocześnie koniec pamięci programu. Zwykle adres AD jest przedstawiony łącznie jako

```
AD=L+256xH
```

lub rozdzielnie jako

```
H=INT(AD/256):L=AD-256xH.
```

Komputer - w czasie pracy - posługuje się tym wskaźnikiem.

Przy funkcji FRE tworzy on różnicę wskaźników zawartych w komórkach 49/50 i 51/52 ograniczających obszar wolnej pamięci. Po załączeniu FRE sygnalizuje w przypadku C-16 - 12275, a w przypadku Plus/4 - 60669 wolnych bajtów. W pierwszej części naszego artykułu przez zmianę wskaźników początku zmiennych, zawartość pamięci zmiennych została sprowadzona na ekran. Pozwaliło to zaobserwować, jak komputer układa w pamięci zmienne proste, w kolejności ich podawania, w pakietach siedmio-bajtowych. W pierwszych dwóch bajtach, w których umieszczona jest nazwa zmiennych, wyróżnia się cztery typy zmiennych. Pozostałe pięć bajtów zawiera w przypadku zmiennych zmienno-przecinkowych i całkowitych różnie zakodowane wartości liczb.

Aby przenieść pamięć zmiennych na ekran, wystarczy starsze bajty początku zmiennych zmienić z 16 (16x256=4095 - początek BASIC'a) na 12(12x256=3072 - początek ekranu).
Można to zrobić podając rozkaz:

```
POKE46,12:CLR.
```

Po co jest potrzebne CLR, przecież żadna zmienna nie powinna być skasowana?

Rozkaz CLR umieszcza ruchomy wskaźnik (komórki 47/48 i 49/50) pod adresem, określonym zawartością komórek 45/46, a więc na początku zmiennych. Ta właściwość jest użyta w celu zaoszczędzenia dwóch POKE'ów. Licznik liczy wolne bajty między wskaźnikiem 49/50 i 51/52 i FRE sygnalizuje 1024 bajtów więcej niż po załączeniu. Naturalnie nie można tego „rozszerzenia pamięci" wykorzystać.

Ponadto nie trzeba się martwić, że ten „POKE" może wyrządzić szkodę komputerowi, za pomocą przycisku RESET można zawsze wrócić do stanu normalnego. Najlepiej jest teraz przełączyć się na pisownię małych liter, aby łatwiej można było odszyfrować kod ekranu i wymazać ekran przyciskiem CLR.

Wpiszmy teraz:

```
FORii=1TO10STEP3:ii%=ii:FORxx=OTO100:NEXTxx,ii
```

W pierwszym momencie niewiele można rozpoznać chociaż użyta w celu zwłoki została pętla xx.

Na końcu w górnej linii bez odstępów umieszczone są trzy grupy siedmiocyfrowe.

Zmienne zmienno-przecinkowe ii i xx rozpoczynają się normalnie wydrukowanymi dużymi literami II i XX, podczas gdy przy zmiennych całkowitych litery ii są drukowane w inwersji.

Dlaczego tak jest, zostało obszernie omówione w pierwszej części.

Powtarzamy teraz przebieg raz jeszcze i obserwujemy tylko zmienną całkowitą w górnej linii. Zmienia się ona na tyle powoli, że można zaobserwować jak w 4 bajcie zmieniają się po kolei małe litery od a do i, podczas gdy inne bajty liczbowe pokazują niezmiennie klamry - kod ekranowy zero, litery a do i - kody od 1 do 9. Za pomocą PRINT ii% można stwierdzić, że zmienna całkowita faktycznie ma wartość 9. Jeśli „puści" się film jeszcze raz i skoncentruje uwagę na pierwszej grupie siedmiocyfrowej, dostrzeże się zmianę wszystkich 5 - liczbowych bajtów. Ponieważ STEP=3, więc przebiega to trzy razy szybciej niż przy zmiennych całkowitych. Przedstawienie liczbowe w przypadku zmiennych całkowitych jest bardzo proste. Liczba zostaje podzielona na młodsze i starsze bajty.

Młodsze bajty, jak można zobaczyć stoją na 4 pozycji grupy siedmiocyfrowej, starsze są zaszyfrowane na pozycji 3, leżą więc przed młodszymi.

Po wyczyszczeniu ekranu przyciskiem CLR wpiszcie teraz większą liczbę: x%=257. Na pozycji 3 i 4 pojawia się „a" o kodzie 1 bo 1x256+1=257

Dla kontroli można zamienić zmienną w pamięci, np. pierwsze „a" na „b", a drugie zamienić na „c".

Po tym x% musi przyjąć wartość 2x256+3=515.

Można to bez większego wysiłku sprawdzić, przez wpisanie w jedną z wolnych linii print x%.

Za pomocą dwóch bajtów można przedstawić 256x256=65536 zmiennych całkowitych. Rozciąga się to na wartości od -32768 przez zero do +32767.

Ujemne zmienne całkowite można rozpoznać po tym, że pierwszy bit starszego bajtu ma wartość 1, czyli starszy bajt ma wartość większą niż 127. Oznacza to w kodzie ekranu, że 3 pozycja jest wydrukowana w inwersji, zresztą kod ujemny liczby całkowitej wygląda zupełnie inaczej niż dodatniej, o tej samej sumie. Trzy ostatnie pozycje grupy siedmiocyfrowej służą tylko do uzupełniania. Można tutaj wprowadzić dowolne oznaczenie i nie wpływa to na wartość zmiennej.

Wskazówka nr 7:
użycie prostych liczb całkowitych, nie wpływa na zaoszczędzenie miejsca w pamięci, ani nie przynosi żadnych oszczędności czasowych ponieważ zostają one zaraz przemienione na format zmienno-przecinkowy.

Co się stanie, gdy zostanie przekroczony dopuszczalny zakres liczb?

```
for    l=0to3:a%=32767+1:printa%:next
```

Po przekroczeniu górnej granicy dwubajtowy licznik przeskakuje z powrotem do najniższej wartości i liczy dalej, tak samo jak licznik kilometrów, który „skacze "z 99999 na 0.

Wskazówka nr 8:
uważaj przy obliczeniach z wysokimi liczbami całkowitymi. Po przekroczeniu wartości 32767 nie jest sygnalizowany błąd. Wada ta nie dotyczy zmiennych zmienno-przecinkowych.

Dla przedstawienia zmiennych zmienno-przecinkowych używa się 5 bajtów. W przypadku skomplikowanego kodowania liczba zostaje najpierw rozdzielona na następną niższą od 2 potęgę i resztę zwaną mantysą. Komputer koduje binarnie mantysę i umieszcza na czterech ostatnich pozycjach.

Drugi wykładnik podwyższony o znormalizowaną wielkość 129 zostaje umieszczony na pozycji 3. Najlepiej można to pokazać na przykładzie:

```
ab=5.4=1.35x22.
```

Trzeci bajt powinien wynieść 2+129=131.

W kodzie ekranu jest to małe e w inwersji (po podaniu ab=5.4 faktycznie na 3 pozycji grupy siedmiocyfrowej pojawia się taka literka).

A jak to jest z liczbami niższymi od 1, których wykładnik jest ujemny.

Przykładowo a=0.5=1/2 - wykładnik równy -1. Na trzeciej pozycji powinno być (-1)+129=128 i na ekranie jest to klamra w inwersji. Drugi wykładnik może przyjmować wartość od -128 do +126. Komputer ma więc zakres liczb od 2.95*10↑39 do 1.7*10↑38.

Następujące prawidłowości dają się szybko dowieść.

1. Przy drugich potęgach wszystkie cztery pozycje mantysy mają wartość zero.
2. Przy podnoszeniu dowolnej liczby do drugiej potęgi zmienia się tylko wykładnik na trzeciej pozycji, a mantysa pozostaje niezmieniona.

Porównując dowolną liczbę z jej ujemnym odpowiednikiem a=3.456 i b=-3.456 widzi się, że jedyna różnica występuje w pozycji 4, w 1-szym bajcie mantysy. Tam umieszczony jest znak liczby.

I znowu jest to tak, że wartości powyżej 127 oznaczają liczbę ujemną i na ekranie przedstawione są w inwersji. Nie tylko dla początkującego wydaje się to kodowanie skomplikowane, ale także komputer na przekształcenie takich liczb potrzebuje sporo czasu. Można zmierzyć ten czas za pomocą wewnętrznego zegara komputera.

Czas wskazywany przez ten zegar jest umieszczony w pamięci od komórki 163 do 165.

Za pomocą rozkazu przemieścimy zegar na ekran.

```
fori=0to1000:poke3072,peek(163):poke3073,peek
(164):poke3074,peek(165):next
```

Pierwszy rozkaz POKE oznacza wpisanie do pamięci (komórka 3072 - a więc w górnym lewym rogu ekranu) zawartości 163 komórki pamięci.

Kiedy teraz pomnożymy lewą liczbę przez 256x256, a środkową przez 256 i oba wyniki dodamy do prawej liczby, wtedy dowiemy się dokładnie ile minęło czasu od włączenia komputera. Pamiętajmy, że przy załączeniu komputera (i po każdym RESET) zegar jest zerowany!

Naturalnie można by szybciej i wygodniej uzyskać wynik za pomocą instrukcji **print ti.**

Wewnętrzny zegar komputera można użyć do pomiaru czasu wykonywania programów czy procedur. Trzeba wtedy np. stukrotnie powtórzyć daną procedurę odczytać czas realizacji i podzielić go przez sto...

Wskazówka nr 9:
pomiar krótkich odstępów czasowych

```
10n=100:ta=ti:fori=1ton
20a=3.14159265+3.14159265
50next:te=ti:print(te-ta)/n/.06,„ms"
```

W linii 10 podano liczbę przebiegów i zachowano czas początkowy ti.

Między liniami 20 i 50 mogą zostać umieszczone dowolne rozkazy i procedury. Czas ogólny (te-to) jest w linii 50 dzielony przez liczbę przebiegów n, chcąc uzyskać czas w milisekundach, dzielimy całe wyrażenie przez 0,06. Aby uzyskać wynik w sekundach wystarczy podzielić przez je 60.

Zanim uruchomimy ten program, komputer powinien być doprowadzony do stanu początkowego przyciskiem RESET. W naszym przykładzie operacja obliczeniowa, której czas trwania chcemy zmierzyć znajduje się w linii 20. Dwie stałe z ośmioma miejscami po przecinku, powinny być dodane. Okazuje się, że C-16 i Plus/4 potrzebuje na to 61 lub 63 ms.

Stara reguła programowania mówi, aby kilkakrotnie występujące stałe zmieniać na jedną zmienną. Dlatego gdy linię 20 zmienimy na:

```
20a=3.14159265:b=a+a
```

to czas tej operacji wyniesie tylko 35 ms.

Zysk czasu jest znaczny! O wiele szybciej można to zrobić, gdy komputer ma w pamięci stałą w formie zmienno-przecinkowej. Wpiszcie:

```
b=pi+pi
```

- potrzeba mniej niż 4ms!

Wskazówka nr 10:
aby zaoszczędzić czas opracowywania, należy występujące wielokrotnie stałe zamienić na zmienne.

Często w programach pisze się „a*a" zamiast a do potęgi 2 (a↑2)

Czy ma to jakieś znaczenie? By o tym się przekonać wprowadzimy do naszego programu nową linię:

```
5a=12.34.
```

W linii 20 wpiszcie najpierw b=a↑2 dokonajcie pomiaru, następnie zmieńcie:

```
b=a*a.
```

C-16 i Plus/4 potrzebuje na a↑2 48,2 ms, a na a*a 6,2 ms!

Wskazówka nr 11:
zastępuj w miarę możliwiści potęgi przez mnożenie, a*a jest siedem razy szybciej obliczone niż a↑2.

Na zakończenie jeszcze jeden program - budzenie, który także używa zegara czasu rzeczywistego.

Ten „jednowierszowiec" ma programowalny czas „budzenia" od 1 sekundy do 24 godzin.

Wskazówka nr 12:
„Budzik".

```
1tw$=„hhmmss": ti$=„000000": do while ti$ tw$:
loop: vol8: sound 1,800,100
```

W tw$ podaje się po ilu godzinach minutach i sekundach budzik powinien zadziałać.

Pętla „do while" biegnie tak długo, aż łańcuch ti$, który jest sterowany wewnętrznym zegarem, będzie większy niż tw$. Potem program skacze do razkazu, który występuje po loop i wyzwala sygnał akustyczny.

Opracowano na podstawie RUN 3/87.

OGŁOSZENIA

NIE ZWLEKAJ
JUŻ DZIŚ ZAMÓW
W KIOSKU RUCHU
GRUDNIOWY NUMER
„64 PLUS 4"!

Sprzedam rozszerzenie pamięci do C-128 (512kB), cena ok. 700tys. zł. Sprzedam FINAL III + instrukcja, cena 250tys. zł. oraz 100 nagranych dysków do C-64. Info wyłącznie listownie. Tomasz Pencherz, ul. Nałkowskiej 2, 42-600 Tarnowskie Góry.

Kupię (wymienię) MAKROASSEMBLER lub ASSEMBLER z opisem lub instrukcją do C-64. Artur Karaźniewicz, 11-111 Kraszewo 33, woj. olsztyńskie.

Poszukuję programów i literatyry - SHARP MZ-731. Bukowski, ul. Moniuszki 11 m 57, 11-400 Kętrzyn.

Tanio sprzedam ATARI 65 XE + XCA-12 + kasety (50 programów). AMIGĘ kupię. Krzysztof Kaniewski, ul. Orkana 18/3, 10-012Olsztyn.

C-64: wymienię kasety. Artur Czajkowski, ul. Wyki7/10, 01-318 Warszawa.

Sprzedam C-64 II + stację VC-1541 II + monitor + magnetofon + final III + top star (5mln). Ciszewski Dariusz, ul. ZWM 40 m 34, 42-200 Częstochowa.

YAKUZA'S PLAYERS CLUB - C-64 zaprasza. Oś. Dywizjonu 303, bl. 41/53, 31-875 Kraków.

Jest to chyba najtańszy z programów emitujących mowę. Można go wykorzystać na wiele sposobów, np. w grach, czy w menu (spisie). Zapewne zdziwił Was realistycznie brzmiący dźwięk w niektórych grach komputerowych. Za pomocą tutaj przedstawionego programu i z wykorzystaniem przetwornika analogowo - cyfrowego, można bezpośrednio wydobyć głos i dowolne dźwięki.

Przetwornik analogowo - cyfrowy jest to urządzenie przekształcające informacje z mikrofonu, radia i magnetofonu na kod cyfrowy zrozumiały przez komputer. Różnica między sygnałem analogowym i cyfrowym stanie się jasna, gdy porówna się zegar wskazówkowy z cyfrowym.

Zegar ze wskazówkami, które obracają się powoli, ale ciągle, przedstawia sygnał analogowy. Im cieńsza jest wskazówka i silniejsze szkiełko powiększające, tym dokładniej możesz odczytać czas.

W przypadku zegara cyfrowego, użycie szkiełka powiększającego nie zda się na nic, ponieważ oznaczenie czasu nie zmienia się ciągle lecz w skokach np. sekundowych. Będziecie teraz na pewno oponować, że dla przekształcenia dźwięku w postać cyfrową, konieczne jest spore oprzyrządowanie dodatkowe. Tak jednak nie jest, ponieważ każdy magnetofon posiada elektronikę, konieczną do przekształcenia sygnału w postać cyfrową. Potrzebujesz więc jedynie C-64 i magnetofon. Lutowanie ogranicza się wyłącznie do przylutowania mikrofonu, lub gniazda mikrofonowego do złącza w magnetofonie.

HURA! ON MÓWI!

DODATKOWE ROZSZERZENIE BASIC'U.

Zanim zabierzesz się do rozebrania magnetofonu, trzeba najpierw wprowadzić program „MOWA" (listing 1) i nagrać go na taśmie. Program ładujemy za pomocą LOAD "MOWA", 8,1. Po starcie SYS 49231 zmieniają się barwy obrazu i zmodyfikowany sygnał zgłoszenia pojawia się na ekranie. Teraz mamy do dyspozycji 5 nowych rozkazów w Basic'u i tylko 2045 bajty wolnego RAM'u. Powodem tego, że do dyspozycji zostało tylko 2045 bajtów jest to, że zapis cyfrowy mowy wymaga dużej ilości pamięci. Dla zapamiętania 13 sekund mowy konieczne jest około 38000 bajtów.

Aby mowa została zapisana w możliwie najwyższej jakości, ekran jest wygaszony podczas procesu przetwarzania i odtwarzania dźwięku, przerwanie przez wideokontroler tego procesu mogłoby mieć negatywny wpływ na jakość dźwięku. W celu uzyskania, jak największej elastyczności po zainstalowaniu programu, pamięć C-64 została podzielona na różne obszary, w których mogą być zapamiętywane słowa lub dźwięki. Pojedyncze przedziały można w programie dowolnie często przywoływać i uszeregowywać.

5 DODATKOWYCH ROZKAZÓW W BASIC'U

Cyfra postawiona za rozkazem, decyduje, którego przedziału RAM'u rozkaz dotyczy. Liczba w opisie rozkazów jest przedstawiona jako >X< i może przyjmować następujące wartości:

X=1 do 16.	Zakres od 1 do 16 jest zarezerwowany dla słów jedno, względnie dwusylabowych. Można tutaj zapamiętać np. liczby lub krótkie wyrazy.
X=17.	Zakres dla zapisu słów kilku sekundowych. Można wpisać krótkie zdania.
X=255.	Jeśli liczba 255 zostanie zapisana za rozkazem, to rozkaz pokrywa zakres od 1 do 17. Zostanie więc zapisana cała pamięć. Możemy przetworzyć około 13 sekund. Wszystkie inne liczby będą interpretowane jako 17.

TERAZ ROZKAZY:

Lx >learn< - po wprowadzeniu tego rozkazu ekran zostaje wygaszony. Sygnały dźwiękowe zostaną przekształcone w postać cyfrową i zapamiętane. >X< decyduje, w którym obszarze pamięci będą umieszczone.

Sx >say< - po tym rozkazie ekran zostanie także wygaszony. Z głośnika telewizora lub monitora rozlegnie się teraz, to co wcześniej nagrałeś. W przciwieństwie do rozkazu >learn< ten rozkaz nie wymaga dodatkowego oprzyrządowania. Może więc być użyty bez dalej opisanej przeróbki. >X< decyduje z jakiego obszaru pamięci będą podawane dźwięki.

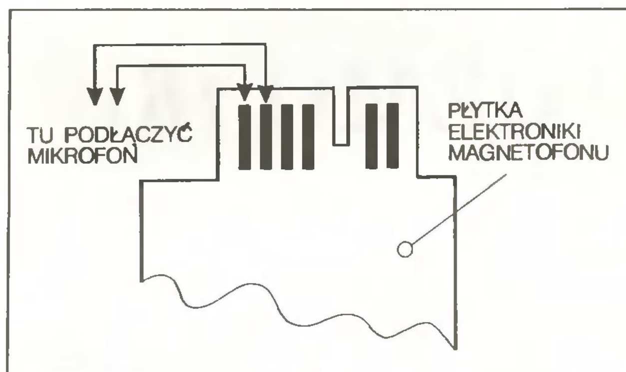
P „nazwa" X >put< za pomocą tego rozkazu można sekwencję dźwięków, które znajdują się w pamięci przegrać na dyskietkę. (13 sekundowy zapis zajmuje 178 bloków), >X< mówi, który zakres będzie zapamiętany.

G nazwa >get< ten rozkaz będzie używany do ładowania sekwencji dźwięków.

R >rate< za pomocą tego rozkazu można wpłynąć na szybkość odtwarzania. Po rozkazie musi nastąpić wyrażenie numeryczne (liczba) określające szybkość. Dozwolone są liczby od 0 do 19. Przy czym 0 odpowiada najniższej, a 19 najwyższej prędkości (13 odpowiada normalnej prędkości mowy).

PRZERÓBKA HARDWAR'u

Jak wyżej wspomniano, do przekształcenia sygnału w postać cyfrową będzie użyty magnetofon. Aby dokonać odpowiedniej przeróbki trzeba go otworzyć. W tym celu należy wykręcić 4 śruby z dołu. Po zdjęciu pokrywy ukaże się płytka ekranująca z folii aluminiowej. Z pod folii wystaje na zewnątrz złącze (rys.1). Przylutuj do tego złącza w oznaczonych punktach mikrofon lub odpowiednie gniazdo. To już cała przeróbka! Jeśli chcesz znowu magnetofon używać jako pamięć masową, to należy mikrofon odłączyć, aby zapobiec zakłóceniom podczas transmisji danych między C-64 i magnetofonem. Miejcie na uwadze, że po otwarciu magnetofonu tracicie gwarancję. Jeśli nie macie doświadczenia w lutowaniu, namawiamy Was, aby przeróbkę wykonać pod okiem fachowca.



Rys.1 - tak podłącza się zwykły mikrofon do magnetofonu.

Listing 1 Program „MOWA".

```
c000 : 93 90 20 20 20 20 20 20   ab
c008 : 2a 2a 2a 20 56 4f 49 43   62
c010 : 45 20 36 34 20 53 59 53   22
c018 : 54 45 4d 20 56 32 2e 30   76
c020 : 20 2a 2a 2a 0d 0d 31 39   96
c028 : 38 35 20 42 59 20 4d 2e   73
c030 : 4b 4c 49 4e 47 45 52 20   e6
c038 : 20 32 30 34 35 20 42 41   e4
c040 : 53 49 43 20 42 59 54 45   d8
c048 : 53 20 46 52 45 45 00 a9   59
c050 : 00 a0 c0 20 1e ab a9 0f   d8
c058 : 8d 21 d0 a9 0b 8d 20 d0   1e
c060 : 4c 42 c2 00 00 00 20 73   e5
c068 : 00 08 c9 5f f0 04 28 4c   33
c070 : e7 a7 28 20 73 00 08 c9   24
c078 : 4c f0 24 c9 53 f0 2b ea   be
c080 : 4c 3a c1 ea 20 9b b7 08   85
c088 : e0 11 90 04 ea 4c ad c1   e1
c090 : ea a9 06 69 09 ca d0 fb   20
c098 : 85 fb 69 09 85 fc 60 28   a8
c0a0 : 20 9b b7 08 20 88 c0 4c   5f
c0a8 : 5d c1 28 20 9b b7 08 20   cc
c0b0 : 88 c0 4c dd c1 28 20 13   6b
cob8 : c1 ea ea ea ea ea ea ea   8e
c0c0 : a2 08 a0 01 20 ba ff a2   cc
c0c8 : 3c a0 03 a5 ff 20 bd ff   c2
c0d0 : a9 00 a2 00 a0 00 20 d5   58
c0d8 : ff ea 4c e4 a7 20 13 c1   47
c0e0 : 28 20 9b b7 08 20 88 c0   1b
c0e8 : a9 36 85 01 a2 08 20 ba   8e
c0f0 : ff a2 3c a0 03 a5 ff 20   01
c0f8 : bd ff a2 00 a4 fb 86 fd   9e
c100 : 84 fe a9 fd a2 01 a4 fc   ec
c108 : 20 d8 ff a9 37 85 01 28   bd
c110 : 4c e7 a7 20 73 00 c9 22   e0
c118 : f0 03 4c 08 af ea a0 00   73
c120 : 20 73 00 c9 22 f0 0b 99   3c
c128 : 3c 03 c8 c0 10 f0 03 4c   5d
c130 : 20 c1 ea 84 ff 60 00 00   7f
c138 : 00 00 c9 47 f0 0c c9 50   cb
c140 : f0 9b c9 52 f0 07 28 4c   3b
c148 : 08 af 4c b5 c0 28 20 9b   f7
c150 : b7 08 e0 14 90 03 4c 48   a9
c158 : b2 4c c3 c1 ea ad 11 d0   5b
c160 : 29 ef 8d 11 d0 a9 7f 8d   7a
c168 : 0d dc 78 a9 00 85 fa a5   9a
c170 : fb 85 fb a0 00 a2 08 ad   d1
c178 : 0d dc 4a 4a 4a 4a 85 fe   da
c180 : 06 ff a5 ff 05 fe 85 ff   4d
c188 : ca d0 ec 91 fa 86 ff 88   1c
c190 : d0 e3 e6 fb a5 fb c5 fc   d6
c198 : d0 d9 ad 11 d0 09 10 8d   93
c1a0 : 11 d0 58 a9 81 8d 0d dc   d7
c1a8 : 28 4c e7 a7 ea e0 ff f0   7d
c1b0 : 09 a9 9f 85 fb a9 c0 85   41
c1b8 : fc 60 a9 10 85 fb a9 c0   b1
c1c0 : 85 fc 60 8a 69 0f 8d 0d   8c
c1c8 : c2 28 4c e7 a7 00 00 00   29
c1d0 : 00 00 00 00 00 00 00 00   d1
c1d8 : 00 00 00 00 00 a9 36 85   0a
c1e0 : 01 a9 7f 8d 0d dc 78 ad   3c
c1e8 : 11 d0 29 ef 8d 11 d0 ea   24
c1f0 : a9 00 85 bb a5 fb 85 bc   3c
c1f8 : a0 00 a2 08 b1 bb 0a 48   f4
c200 : 90 0a a9 0f 8d 18 d4 a9   22
c208 : 00 8d 18 d4 4c 1c c2 ea   f6
c210 : ea ea ea ea ea ea ea ea   0f
c218 : ea ea ea ea ea ea ea ea   17
c220 : ea ea 68 ca d0 d8 88 d0   8a
c228 : d1 e6 bc a5 bc c5 fc d0   e0
c230 : c7 ad 11 d0 09 10 8d 11   96
c238 : d0 58 a9 37 85 01 ea 4c   2a
c240 : a3 c1 a9 66 8d 08 03 a9   73
c248 : c0 8d 09 03 a9 10 85 34   0b
c250 : 85 38 20 44 a6 4c 74 a4   6a
c258 : 00 45 00 fb 00 fb 00 fb   52
```

UWAGA! Ostatnia, oddzielona kolumna liczb jest sumą kontrolną danego wiersza.

Opr. H.S.

JAK ZROBIĆ DEMO (8)

Po lekturze poprzedniego odcinka naszego cyklu możemy chyba uznać problem wykonania prostego jednoznakowego scroll'a za zamknięty. Nieco inaczej ma się sprawa z płynnym przesuwaniem tekstów o innym formacie czcionek, a już na pewno czymś nowym będzie wykonanie przesuwania tekstu do góry, czyli taka technika, którą najczęściej możemy zaobserwować na zakończeniu każdego filmu, gdzie wyświetlane są w ten sposób nazwiska aktorów, reżysera, itp.

Zasada płynnego przesuwu pozostaje taka sama, tzn. o wysokość znaku przesunięcie dokonuje się drogą sprzętową (komórka $D011), po czym następuje przesunięcie całego ekranu w górę o jeden wiersz, druk następnej linii i cały cykl wykonywany jest od początku. Główna różnica polega na tym, że nowa linia pojawia się jako 40 kolejnych znaków, co odpowiada szerokości ekranu.

Dla osób wprawionych w śledzeniu zamieszczanych w tym dziale listingów nie będzie problemem wychwycenie wszystkich niuansów.

Tekst programu jak zwykle napisano przy użyciu Turbo Assemblera.

```
        *=  $1000        ;adres startu assemblacji

        LDA   #$01
        STA   $0286      ;ustalanie atrybutow koloru
                         ;tla
        STA   $D021
        JSR   $E544      ;kasowanie   ekranu
        SEI
        LDA   #$7F
        STA   $DC0D      ;ustawianie IRQ-
                         ;graficznego
        LDX   #$00
        STX   $DC0E
        STX   $D020
        STX   $D021
        INX
        STX   $D01A
        LDA   #$1B
        STA   $D011
        LDA   #$F9
        STA   $D012      ;przerwanie w linii $F9
        LDA   #q
        LDY   #irq
        STA   $0314
        STY   $0315      ;zmiana  wektora  IRQ
        JSR   scrlset    ;ustawienie parametrow
        CLI
loop    JMP   loop       ;petla (do wykorzystania)

irq     JSR   scroll
        INC   $D019
        JMP   $EA7E      ;zakonczenie   IRQ
upbyte  .BYTE $00        ;bajt  do  zapisu  wartosci
                         ;$D011

scrlset LDA   #$17
        STA   upbyte
        LDA   #$01
        STA   scroll+1   ;ustawienie    licznika
txtset  LDA   #          ;ustalenie  adresu  tekstu
                         ;w pamieci
        STA   $20
        LDA   #>txt
        STA   $21
        RTS

scroll  LDA   #$00       ;wartosc    licznika
        BEQ   scr1
        DEC   scroll+1
        RTS
scr1    LDA   #$01
        STA   scroll+1   ;zaladowanie    licznika
        LDA   upbyte     ;obsluga scroll'a w gore
        SEC
        SBC   #$01
        CMP   #$10
        BCS   scr2
        LDA   #$17
scr2    STA   upbyte
        STA   $D011
        CMP   #$17
        BEQ   scr3
        RTS
scr3    JSR   takeup     ;przsuniecie o jeden
                         ;wiersz  w gore
        LDY   #$00
scr5    LDA   ($20),Y
        STA   $07C0,Y
        BEQ   scr4       ;zero? TAK-koniec tekstu
        INY
        CPY   #$28
        BCC   scr5
        BCS   scr6
scr4    JSR   txtset     ;tekst  od  poczatku
        BNE   scr3+3
scr6    TAY
        CLC
        ADC   $20
        STA   $20
        BCC   scr7
        INC   $21
scr7    RTS

takeup  LDX   #$27       ;szybki scroll w gore
tp1     LDA   $0428,X
        STA   $0400,X
        LDA   $0450,X
        STA   $0428,X
        LDA   $0478,X
        STA   $0450,X
        LDA   $04A0,X
        STA   $0478,X
        LDA   $04C8,X
        STA   $04A0,X
        LDA   $04F0,X
        STA   $04C8,X
```

```
         LDA    $0518,X
         STA    $04F0,X
         DEX
         BPL    tp1
         LDX    #$27
tp2      LDA    $0540,X
         STA    $0518,X
         LDA    $0568,X
         STA    $0540,X
         LDA    $0590,X
         STA    $0568,X
         LDA    $05B8,X
         STA    $0590,X
         LDA    $05E0,X
         STA    $05B8,X
         LDA    $0608,X
         STA    $05E0,X
         DEX
         BPL    tp2
         LDX    #$27
tp3      LDA    $0630,X
         STA    $0608,X
         LDA    $0658,X
         STA    $0630,X
         LDA    $0680,X
         STA    $0658,X
         LDA    $06A8,X
         STA    $0680,X
         LDA    $06D0,X
         STA    $06A8,X
         LDA    $06F8,X
         STA    $06D0,X
         DEX
         BPL    tp3
         LDX    #$27
tp4      LDA    $0720,X
         STA    $06F8,X
         LDA    $0748,X
         STA    $0720,X
         LDA    $0770,X
         STA    $0748,X
         LDA    $0798,X
         STA    $0770,X
         LDA    $07C0,X
         STA    $0798,X
         DEX
         BPL    tp4
         RTS
txt      = $A200              ;przykładowy adres tekstu
```

Wnikliwi obserwatorzy zapewne zauważyli, że procedura o nazwie „TAKEUP" została napisana w nieco „długi" sposób. Forma ta nie grzeszy krótkością, ale za to jest jedną z niewątpliwie najszybszych (a już na pewno jest szybsza niż np. podobna, napisana z wykorzystaniem adresowania przez stronę zerową pamięci) co ma znaczenie w przypadku, gdy zechcemy ją wykorzystać razem z innymi procedurami.

W naszym przykładzie adres tekstu ustaliliśmy na $A200 tylko po to, by można było szybko i wygodnie sprawdzić, czy całość sprawnie pracuje (w tym miejscu pamięci znajdują się w ROM teksty komunikatów j. BASIC). W konkretnym zastosowaniu wykorzystaniu tekst należy umieścić w pamięci w postaci kodów ekranowych, przy czym kod zero jest znacznikiem końca tekstu. Właściwy adres startu należy w takim przypadku umieścić, modyfikując adresy w procedurze o nazwie „TXTSET".

Paweł Sołtysiński

Programy ciekawe, zwariowane i takie sobie

Witajcie!

Tym razem programów do naszej zwariowanej rubryki przyszło tyle, że na biurku utworzył mi się pokaźny stosik listów. Ze wszystkich wybrałem dwa, które przedstawiam poniżej.

Pewcł Tutka z Gliwic przysłał do nas programik sprawdzający stan bitów w bajcie.

```
10  PRINT „PODAJ LICZBE";: INPUT A: PRINT
20  PRINT „PODAJ ILOSC BAJTOW";: INPUT B
30  B=B*8-1
40  FOR Q=B TO 0 STEP -1:Z(Q)=0
50  IF A=2↑Q THEN A=A-2↑Q: Z(Q)=1
60  NEXT Q
70  PRINT
80  FOR Q=B TO 0 STEP -1: PRINT Z(Q);: NEXT Q
```

Teraz czekam na programik zapalający dowolny bit w bajcie!

Jacek Zaczko ze Szczecina przysłał dwa programy do naszej rubryki. Do druku wybrałem „Płynne przesuwanie napisów".

Oto sam program:

```
10  DIM G$ (14):ILOSC WYRAZOW
15  FOR W=0 TO 19: READ G$(W)
20  PRINT CHR$(147)
25  FOR A=39 TO 0 STEP -1
30  PRINT TAB(A) "";G$(W);PRINT CHR$(147)
35  NEXT A
40  NEXT W
100 DATA     COMMODORE,64,+,4,AMIGA,
    PRZEDSTAWIA
110 DATA     PROGRAM,DEMONSTRACYJNY,
    PRZESUWANIE
120 DATA     TEKSTOW,AUTOR,:,JACEK,ZACZKO
```

Jest to jedno z możliwych rozwiązań. Program przesuwa jednak całe wyrazy a nie poszczególne litery. Jest to wada tej koncepcji.

Czekam na inne pomysły realizacji tej procedury. Najciekawsze wydrukujemy.

Sambor Kuźma

Coś do „wklepania", czyli mikro - gra

Chcąc zrobić niespodziankę wszystkim tym, którzy lubią przepisywanie programów z magazynów komputerowych, postanowiłem napisać coś, co mimo tego, że bardzo krótkie, będzie już godne nazwy gry.

Zasady są bardzo proste: joystick'iem podłączonym w porcie numer 2 sterujemy naszym pojazdem, którym musimy bezkolizyjnie przejechać po krętych górskich drogach. Biorąc pod uwagę fakt, że aktualnie zjeżdżamy z góry, należy uznać za normalne, że prędkość ulega zwiększeniu z każdą chwilą. Kto będzie w stanie dłużej prowadzić swój pojazd, ten uzyska więcej punktów, przy czym najlepszy dotychczasowy wynik jest automatycznie zapamiętywany, jako tzw. HIGH - SCORE.

Jeżeli chodzi o stronę konstrukcyjną programu, to zasadniczo składa się on z dwóch części:

- programu w języku BASIC
- procedur w kodzie maszynowym,

dzięki którym cała animacja odbywa się naprawdę szybko. Także sterowanie naszym pojazdem odbywa się z poziomu języka maszynowego. Teraz wystarczy już tylko dokładnie przepisać zamieszczony poniżej listing i rozpocząć grę. Przy okazji chciałbym nadmienić, że mój własny rekord wynosi 92 punkty!

Paweł „POLONUS" Sołtysiński

```
0     REM**(C)Pawel   Soltysinski**
1     REM**C-64V.(C-64+4  ■  AMIGA)**
2     :
3     :GOSUB  100
4     :
10    POKE53280,0:POKE53281,0:C=20649:PRINTCHR$(5);CHR$(147);CHR$(14);CHR$(8);
20    PRINT"SCORE:0";TAB(14);"LEVEL:1"
21    PRINTTAB(26);"MOUNTAINRAID":L=1
22    PRINTTAB(26);"BYPOLONUS"
25    PRINT:PRINTTAB(26);"HISCR:";HS
30    FORT=1TO25:SYS20480:NEXT:X=0:SC=0
31    IFPEEK(1584+X)=32THENPOKE832,X+2:POKE830,X+2:GOTO33
32    X=X+1:GOTO31
33    SYS20666:PRINTCHR$(19);:FORT=1TO23:PRINT:NEXT:A$=CHR$(19):EP=833
34    PRINTTAB(6);CHR$(18);"PRESSFIRE"
35    IFPEEK(56320)AND18GOTO35
40    LV=(10-L)*10:PRINTA$;TAB(20);STR$(L):CYT=50
50    PRINTA$;TAB(8);INT(SC):SC=SC+.1:SYSC:FORT=0TOLV:NEXT
51    CYT=CYT-1:IF  CYT=0  AND  L<10  THEN  L=L+1:GOTO40
52    IF  PEEK(EP)=0  THEN  50
53    PRINTA$:FORT=1TO6:PRINT:NEXT:PRINTTAB(6);CHR$(18);"-----------"
54    PRINTTAB(6);CHR$(18);"-GAMEOVER!-":PRINTTAB(6);CHR$(18);"-----------"
55    FORT=1TO255:POKE53280,T:NEXT:SC=INT(SC):IFSC>HSTHENHS=SC
56    GOTO10
98    :
99    REM  **  KOD  MASZYNOWY:  **
100   FOR  T=20480  TO  20728:READ  C:POKE  T,C:NEXT:RETURN
110   DATA   169,152,133,251,133,32,189,192,133,253,133,34,169,7,133,33,133
120   DATA   35,169,219,133,254,133,252,162,23,160,24,177,32,145,34,177
130   DATA   251,145,253,136,16,245,165,32,56,233,40,133,32,133,251,176,4
140   DATA   198,33,198,252,165,34,56,233,40,133,34,133,253,178,4,198,35
150   DATA   198,254,202,208,210,173,19,208,45,18,208,77,2,222,41,3,240
160   DATA   8,201,3,240,4,201,1,176,6,206,63,3,76,102,30,238,63
170   DATA   3,173,63,3,201,2,176,5,169,2,141,83,3,201,17,144,5
180   DATA   169,17,141,63,3,162,24,169,95,157,40,4,169,11,157,40,218
190   DATA   202,16,243,174,63,3,189,1,157,40,216,232,160,5,169,32,157
200   DATA   40,4,169,1,157,40,216,232,136,208,242,169,1,157,40,218,96
210   DATA   174,62,3,169,32,157,48,6,173,64,3,141,62,3,32,0,80
220   DATA   174,64,3,169,31,141,24,212,189,48,6,56,233,32,141,65,3
230   DATA   169,30,157,48,6,169,0,141,24,212,173,0,220,141,255,3,41
240   DATA   4,208,8,173,64,3,240,3,206,64,3,173,255,3,41,8,208
250   DATA   10,173,64,3,201,24,176,3,238,64,3,96
READY
```

ASSEMBLER 6510 - lekcja 5

Po przerwie, w czasie której mam nadzieję nie próżnowaliście tylko sami próbowaliście poznawać kolejne rozkazy mikroprocesora, zabieramy się do dalszej pracy. Dziś poznamy rozkazy pozwalające na operację na bitach i tzw. przesunięcia.

AND - iloczyn logiczny „i"

N	V	D	i	Z	C
*	-	-	-	*	-

Rozkaz	Kod	Cykle
AND #$nn	29	2
AND $nn	25	3
AND $nn,X	35	4
AND $nnnn	2D	4
AND $nnnn,X	3D	4+1
AND $nnnn,Y	39	4+1
AND ($nn,X)	21	6
AND ($nn),Y	31	5+1

Cóż to jest ten iloczyn logiczny? Jest to operacja na bitach w dwu bajtach. Jeśli oba bity są zapalone to w wyniku otrzymujemy 1. W każdym innym przypadku wynikiem jest 0. AND po angielsku znaczy i, łatwo jest więc zapamiętać tę operację np. tak: „jeśli I jeden I drugi bit jest równy 1 to wynik jest też równy 1". Wyobraźmy sobie następujący układ bitów i przeprowadźmy na nim operację AND:

```
        10100110  =  $A6
AND     01100011  =  $63
        --------
        00100010  =  $22
```

Wystarczy sprawdzić teraz na komputerze:

```
1000 LDA #$A6    ; ładowanie do akumulatora A6
1002 AND #$63    ; iloczyn logiczny z wartością 63
1004 BRK         ; koniec
```

W wyniku, w akumulatorze, otrzymujemy spodziewaną wartość 22. Zgadza się?

ORA (or with accumulator) - suma logiczna „lub"

N	V	D	I	Z	C
*	-	-	-	*	-

Rozkaz	Kod	Cykle
ORA #$nn	09	2
ORA $nn	05	3
ORA $nn,X	15	4
ORA $nnnn	0D	4
ORA $nnnn,X	1D	4+1
ORA $nnnn,Y	19	4+1
ORA ($nn,X)	01	6
ORA ($nn),Y	11	5+1

ORA również operuje na bitach tyle, że aby otrzymać „jedynkę" wystarczy aby jeden z nich był „jedynką". Najlepiej zilustruje to przykład:

```
        10100110  =  $A6
ORA     01100011  =  $63
        --------
        11100111  =  $E7
```

```
1000  LDA  #$A6
1002  ORA  #$63
1004  BRK              - w wyniku AC=$E7
```

EOR (exlusive - OR) - suma modulo dwa

N	V	D	i	Z	C
*	-	-	-	*	-

Rozkaz	Kod	Cykle
EOR #$nn	49	2
EOR $nn	45	3
EOR $nn,X	55	4
EOR $nnnn	4D	4
EOR $nnnn,X	5D	4+1
EOR $nnnn,Y	59	4+1
EOR ($nn,X)	41	6
EOR ($nn),Y	51	5+1

Aby zrozumieć, jak działa EOR najlepszy będzie kolejny przykład:

```
        10100110  =  $A6
        01100011  =  $63
        --------
        11000101  =  $C5
```

```
1000  LDA  #$A6
1002  EOR  #$63
1004  BRK              - AC=$C5
```

Z tego przykładu widać, że jeśli bity są „różnowartościowe" to w wyniku otrzymujemy „jedynkę". W innym przypadku wynikiem zawsze będzie „zero".

ASL (arithmetic shift left) - przesunięcie w lewo

N	V	D	I	Z	C
*	-	-	-	*	*

Rozkaz	Kod	Cykle
ASL	0A	2
ASL $nn	06	5
ASL $nn,X	16	6
ASL $nnnn	0E	6
ASL $nnnn,X	1E	7

Rozkaz ten powoduje przesunięcie bitów o jeden w lewo. Do przesuwanego bajtu „wpada" 0 natomiast wysuwany bit znajduje się w C. Oto przykład:

```
         10100110   =  $A6
po ASL
         01001100   =  $4C i 'C'=1
```

Wykonajmy:

```
1000  LDA  #$A6
1002  ASL
1003  BRK
```

W wyniku otrzymujemy w akumulatorze wartość 4C i dodatkowo carry=1.

LSR (logical shift right) - przesunięcie w prawo

N	V	D	I	Z	C
0	-	-	-	*	*

Rozkaz	Kod	Cykle
LSR	4A	2
LSR $nn	46	5
LSR $nn,X	56	6
LSR $nnnn	4E	6
LSR $nnnn,X	5E	7

Rozkaz działa podobnie jak poprzedni tyle, że bity są przesuwane w prawo.

Sambor Kuźma

CZY 128D = 128D?
CZYTAJ
W NASTĘPNYM
NUMERZE!

GRAJ ABY WYGRAĆ

R-TYPE

Po wgraniu gry zresetuj komputer i wpisz następujące komendy:
Poke 12865,173:Poke12957,173 - nieśmiertelność
Poke 12700,96 - odporność na zderzenia
Sys 32768 - ponowny start gry

MENACE

Początek jak poprzednio.
Poke 8980,234:Poke 8981,234 - energia
Poke 8228,0 - pociski do działa
Poke 8243,0:Poke 8261,0 - Energia do laserów
Sys 2080 - restart gry

THUNDERBLADE

Początek jak poprzednio.
Poke 4159,255 - 255 helikopterów
Sys 4096 - restart gry.

EMPIRE STRIKE BACK

By uzyskać nieskończoną ilość energii w czasie gry wciśnij spację oraz B,N,M,J,K,L.

Robert 'Mr.Raf' Turliński

WSZYSTKICH ZAINTERESOWANYCH NABYCIEM ZALEGŁYCH NUMERÓW

„64 plus 4 & AMIGA”

INFORMUJEMY, ŻE POSIADAMY JESZCZE OGRANICZONĄ ILOŚĆ NUMERÓW

OD LISTOPADA 1990R. DO WRZEŚNIA 1991R.

ZAMÓWIENIA PROSIMY KIEROWAĆ NA ADRES:
Przedsiębiorstwo ABUK sp. z o.o.,
87-200 Wąbrzeźno,
ul. 1 Maja 33.

(Pod tym adresem mieści się dział kolportażu - tam też prosimy przesyłać wszelką korespondencję dotycząca kolportażu czasopisma, dyskietek, taśm itd. Adres redakcji się nie zmienił - patrz stopka.)

ZAMÓWIONE NUMERY PRZEŚLEMY ZA ZALICZENIEM POCZTOWYM.

W związku z pojawiającymi się kłopotami w dystrybucji oferowanych przez nas dyskietek i taśm (wynikającymi z nieczytelnego bądź niekompletnego wypełnienia blankietów wpłat) przedstawiamy obok specjalny druk. Blankiet ten może służyć jako zamówienie i dowód wpłaty dla wszystkich oferowanych przez nas usług: sprzdaż dyskietek i taśm PDP, Voicetracker'a, zamówienie ogłoszeń itd.

REDAKCJA

UWAGA!

Informujemy, że od 1 stycznia 1992r cena 1 egzemplarza „64 plus 4” w prenumeracie wynosi 10.000zł.

Przypominamy, że prenumeratę można zawrzeć na okres nie krótszy niż trzy miesiące DO KOŃCA ROKU KALENDARZOWEGO, tzn do końca 1992r..
Na **CZYTELNIE** wypełnionych blankietach wpłat prosimy dopisać **„PRENUMERATA” oraz okres jej trwania.**
Wpłaty prosimy przesyłać wykorzystując zamieszczony obok blankiet.
Dla wszystkich, którzy zdecydują się wykupić prenumeratę roczną - specjalna okazja - patrz 2 str.

Odcinek dla wpłacającego
Zł ....
słownie ....
wpłacający ....
(dokładny i CZYTELNY adres)
na rachunek:
**Przedsiębiorstwa ABUK sp. z o.o.
87-200 Wąbrzeźno, ul. 1 Maja 33,
Bank PKO SA Bydgoszcz, konto:
5.09011-400522.7-136-11-111.0.**
Opłata
zł........

Odcinek dla Banku
Zł ....
słownie ....
wpłacający ....
(dokładny i CZYTELNY adres)
na rachunek:
**Przedsiębiorstwa ABUK sp. z o.o.
87-200 Wąbrzeźno, ul. 1 Maja 33,
Bank PKO SA Bydgoszcz, konto:
5.09011-400522.7-136-11-111.0.**
Opłata
zł........

Odcinek dla posiadacza rachunku
Zł ....
słownie ....
wpłacający ....
(dokładny i CZYTELNY adres)
na rachunek:
**Przedsiębiorstwa ABUK sp. z o.o.
87-200 Wąbrzeźno, ul. 1 Maja 33,
Bank PKO SA Bydgoszcz, konto:
5.09011-400522.7-136-11-111.0.**
Opłata
zł........

Odcinek dla Poczty
Zł ....
słownie ....
wpłacający ....
(dokładny i CZYTELNY adres)
na rachunek:
**Przedsiębiorstwa ABUK sp. z o.o.
87-200 Wąbrzeźno, ul. 1 Maja 33,
Bank PKO SA Bydgoszcz, konto:
5.09011-400522.7-136-11-111.0.**
Opłata
zł........

TREŚĆ ZAMÓWIENIA:

TREŚĆ ZAMÓWIENIA:

Prosimy o CZYTELNE wypełnienie.

TREŚĆ ZAMÓWIENIA:

Prosimy o CZYTELNE wypełnienie.

TREŚĆ ZAMÓWIENIA:

Prosimy o CZYTELNE wypełnienie.

W związku ze znacznym wzrostem opłat za przesyłki pocztowe, mając na uwadze dobro naszych czyteiników sugerujemy zamawianie numerów zaległych naszego czasopisma na nieco Innych zasadach.

Koszty przesyłki za tzw. zaliczeniu pocztowym przedstawia tabela (rubryki 2, 3, i 4). Wynika z niej, że koszty przesłania dwóch pierwszych numerów „64 plus 4" są większe niż ich wartość! Zamawiając numery wartości 6.000 zł zmuszeni jesteście dopłacić poczcie jeszcze 8.000 zł!

Zupełnie inaczej przedstawia się sytuacja w tabelach 5, 6 i 7. Wpłata kwoty z rubryki nr 7 na nasze konto (wraz z czytelną adnotacją, których numerów dotyczy, umieszczoną na wszystkich odcinkach blankietu) powoduje, że otrzymujecie przesyłkę bez kosztów pobrania!

Proponujemy abyście zaległe numery naszego pisma zamawiali w sposób następujący: wykorzystując tabelę - rubryki 1, 2, 5 i 7 - wyliczyli kwotę wpłaty, a następnie przesłali ją na nasze konto. Po otrzymaniu wpłaty natychmiast realizujemy przesyłkę!

Uwaga: dla dokonania wpłaty prosimy wykorzystać blankiet zamieszczony na tej stronie. Wszystkie dane prosimy pisać czytelnie i - zgodnie z rubrykami na blankiecie - **dziękujemy!**

Numery	Za pobraniem			Wpłata na konto		
	Wart. pisma	Koszt wysyłki	Razem	Wart. pisma	Porto	Razem
1	2	3	4	5	6	7
Xi,XiI	6.000,-	8.000,-	14.000,-	6.000,-	1.000,-	7.000,-
Xi,Xii,i	11.000,-	8.500,-	19.500,-	11.000, 1.500,-	12.500,-	
Xi,XII,i,ii	16.000,-	8.500,-	24.500,-	16.000,-	1.500,-	17.500,-
Xi,XiI,i,ii,iii	21.000,-	8.500,-	29.500,-	21.000,-	1.500,-	22.500,-
XI,Xii,I-iV	26.000,-	10.000,-	36.000,-	26.000,-	2.000,-	28.000,-
Xi,XII,i-V	31.000,-	10.000,-	41.000,-	31.000,-	2.000,-	33.000,-
Xi,XII,i-VI	36.000,-	10.000,-	46.000,-	36.000,-	2.000,-	38.000,-
Xi,XIi +I-Viii	46.000,-	14.000,-	60.000,-	46.000,-	3.000,-	49.000,-

TWORZYMY WŁASNĄ BIBLIOTEKĘ

Często posiadamy wiele różnych procedur, które wykorzystujemy w swoich programach, a które to procedury są uniwersalne dla wielu z nich, np. FileRequester'y, depacker'y i decruncher'y, różne standardowe okienka i wiele, wiele innych. Aby nie wydłużać naszych programów przez włączanie tych samych procedur do nich powinniśmy sobie utworzyć własną bibliotekę! Jak to zrobić?

Otóż nic prostszego: potrzebujemy tylko podstawową wiedzę o programowaniu na Amidze, jakiś asembler (GenAm, ArgAsm, AsmOne czy choćby stara wysłużona Seka), oraz ten artykuł, a także bardzo dużo pomysłów na procedury.

Najlepszą metodą na napisanie własnej biblioteki jest wnikliwa analiza przedstawionego poniżej programu źródłowego, który stanowi przykładową bibliotekę z jedną procedurą - ChangeLED (adres -$1e od bazy biblioteki), włączającą lub wyłączającą filtry.

Wielu własnych bibliotek i aby stały się one standardami tak jak arp, req, czy powerpacker, życzy autor artykułu.

```
*******************************
* Project:  user.library      *
* Coded:  Marcin „Duddie" Dudar *
* Version:                    *
* v1.0  1991, 2 Nov           *
*******************************
Exec            =   4

;exec.library

FreeMem:        =   -210
Remove:         =   -252

LIBF_SUMMING    =   1
LIBF_CHANGED    =   2
LIBF_SUMUSED    =   4
LIBF_DELEXP     =   8

LIBB_SUMMING    =   0
LIBB_CHANGED    =   1
LIBB_SUMUSED    =   2
LIBB_DELEXP     =   3

LIB_FLAGS       =   $e
LIB_PAD         =   $f
LIB_NEGSIZE     =   $10
LIB_POSSIZE     =   $12
LIB_VERSION     =   $14
LIB_REVISION    =   $16
LIB_IDSTRING    =   $18
LIB_SUM         =   $1c
LIB_OPENCNT     =   $20
LIB_SIZE        =   $22

LN_NAME         =   $a
```

```
; dane o naszej bibliotece

Version         =   1
Revision        =   0
Pri             =   0

        ---STRUKTURA BAZY NASZEJ BIBLIOTEKI---

; Moze byc dowolnie modyfikowana jednak musza
w niej wystepowac te zmienne

ul_SegList      =   $22
ul_Flags        =   $26
ul_Pad          =   $27
ul_SIZEOF       =   $28

Start:
        moveq   #0,d0           ;tutaj zaczyna się nasza
                                ;biblioteka

        rts
Resident:
        dc.w    $4afc           ;RTC_MATCHWORD
        dc.l    Resident
        dc.l    CodeEnd
        dc.b    $80
        dc.b    Version
        dc.b    $09
        dc.b    Pri
        dc.l    LibName
        dc.l    IDString
        dc.l    Init

LibName:
        dc.b    'user.library',0    ; nazwa biblioteki
IDString:
        dc.b    'user.library v1.0 (2 Nov, 1991)',13,10,0
        dc.b    '(C)1991 Marcin „Duddie" Dudar',0

CodeEnd:

;tablica inicjujaca
Init:
        dc.l    ul_SIZEOF
        dc.l    FuncTable
        dc.l    DataTable
        dc.l    InitRoutine

        ---TABLICA ADRESOW PROCEDUR---

FuncTable:
        dc.l    ul_Open
        dc.l    ul_Close
        dc.l    ul_Expunge
        dc.l    ul_Null         ;tych czterech adresow
                                ;procedur nie mozna
                                ;zmieniac ani zamieniac
```

```
                                ;miejscami (grozi to blednym
                                ;dzialaniem biblioteki)
        dc.l    ChangeLED
; tutaj adresy naszych procedur
        dc.l    -1

    ---DANE DEFINIUJACE NASZA BIBLIOTEKE---

DataTable:
        dc.b    $e0
        dc.b    $00
        dc.w    $8
        dc.b    9
        dc.b    0
        dc.b    $e0
        dc.b    $00
        dc.w    $9
        dc.b    0
        dc.b    0

        dc.b    $c0
        dc.b    0
        dc.w    LN_NAME
        dc.l    LibName

        dc.b    $e0
        dc.b    $00
        dc.w    LIB_FLAGS
        dc.b    LIBF_SUMUSED+LIBF_CHANGED
        dc.b    0

        dc.b    $d0
        dc.b    $00
        dc.w    LIB_VERSION
        dc.w    Version

        dc.b    $d0
        dc.b    $00
        dc.w    LIB_REVISION
        dc.w    REVISION

        dc.b    $c0
        dc.b    $00
        dc.w    LIB_IDSTRING
        dc.l    IDString

        dc.l    0
-------------------------------------------------------------------------
InitRoutine:                            ;procedura inicjacji
                                        ;biblioteki
    move.l  a5,-(sp)
    move.l  d0,a5
    move.l  a0,ul_SegList(a5)
    ;tutaj moga znajdowac sie procedury np. otworzenie
    ;pozostalych bibliotek i wpisanie adresow do Bazy
    ;naszej biblioteki czy allokacja pewnych obszarow
    ;roboczych na czas pracy naszej biblioteki.

    move.l  a5,d0
    move.l  (sp)+,a5
    rts

        ---STANDARDOWE PROCEDURY---
ul_Open:
    addq.w#1,LIB_OPENCNT(a6)            ;procedura
                                        ;wywolywana za
                                        ;kazdym razem
                                        ;gdy jest otwierana
                                        ;biblioteka
    bclr    #LIBB_DELEXP,ul_Flags(a6)
    move.l  a6,d0
    rts
-------------------------------------------------------------------------
ul_Close:
; procedura wywolywana gdy jest wywolywana procedura
CloseLibrary() i gdy ilosc otworzen biblioteki jest rowna
zero to biblioteka zostanie usunieta (procedura Expunge)
    clr.l   d0
    subq.w  #1,LIB_OPENCNT(a6)
    bne.s   ULCL_End
    btst    #LIBB_DELEXP,ul_Flags(a6)
    bne.s   ul_Expunge
ULCL_End:
    rts
-------------------------------------------------------------------------

ul_Expunge:
    movem.l d1/a5-a6,-(sp)
    move.l  a6,a5
    move.l  Exec.w,a6
    tst.w   LIB_OPENCNT(a5)
    beq.s   ULEX_Remove
    bset    #LIBB_DELEXP,ul_Flags(a5)
    clr.l   d0
    bra.s   ULEX_End
ULEX_Remove:
    move.l  ul_SegList(a5),d2
    move.l  a5,a1
    jsr     Remove(a6)
    clr.l   d0
    move.l  a5,a1
    move.w  LIB_NEGSIZE(a5),d0
    sub.l   d0,a1
    add.w   LIB_POSSIZE(a5),d0
    jsr     FreeMem(a6)
    move.l  d2,d0
ULEX_End:
    movem.l (sp)+,a1/a5-a6
    rts
-------------------------------------------------------------------------

ul_Null:
    moveq   #0,d0                       ;procedura
                                        ;nic nie robiaca
                                        ;jednak jest to
    rts                                 ;czesc
                                        ;skladowa kazdej
                                        ;biblioteki

        ---PROCEDURY WLASNE---

ChangeLED:                              ; nasza procedura
    bchg    #1,$bfe001                  ; zmiana filtrow
    rts

-------------------------------------------------------------------------
```

UWAGA!Procedury Expunge (usuń), Open (otwórz), Close (zamknij) oraz Null (nic) są stałymi składowymi każdej biblioteki i nie mogą być pominięte!!!

Marcin „Duddie" Dudar

KĄCIK POCZĄTKUJĄCEGO KODERA (cz.8)

W dzisiejszym „Kąciku..." poznamy algorytm sortowania bąbelkowego, oraz jak zwykle garść nowych rozkazów.

■ **Sxx <adres efektywny> „Set Acording to Condition"" - ustawienie w zależności od warunku.**

Instrukcja ta testuje warunek określony polem „xx", następnie w zależności od otrzymanego wyniku modyfikuje bajt określony adresem efektywnym. Jeżeli warunek zostanie spełniony wspomniany bajt będzie posiadać wartość -1 ($ff), w przeciwnym przypadku zostanie skasowany (ustawiony na 0).

Pole „xx" posiada analogiczne znaczenie, jak w przypadku instrukcji skoków warunkowych i możliwe są następujące przypadki:

```
SCC - ustawienie,  jeżeli C=0
SCS - ustawienie,  jeżeli C=1
SNE - ustawienie,  jeżeli Z=0
SEQ - ustawienie,  jeżeli Z=1
SPL - ustawienie,  jeżeli N=0
SMI - ustawienie,  jeżeli N=1
SVC - ustawienie,  jeżeli V=0
SVS - ustawienie,  jeżeli V=1
SGE - ustawienie,  jeżeli [N=0 i V=0] albo [N=1 i V=1]
SGT - ustawienie,  jeżeli [N=1 i V=1 i Z=0] albo
                          [N=0 i V=0 i Z=0]
SHI - ustawienie,  jeżeli C=0 i Z=0
SLS - ustawienie,  jeżeli C=1 albo Z=1
SLE - ustawienie,  jeżeli Z=1 lub [N=1 i V=0] lub
                          [N=0 i V=1]
SLT - ustawienie,  jeżeli [N=1 i V=0] lub [N=0 i V=1]
SF  - ustawienie nigdy nie zostanie zrealizowane
ST  - ustawienie wystąpi zawsze bez względu na znaczniki
```

Znaczniki:
Nie są modyfikowane.

Przykład:

SMI $40000 - jeżeli ustawiony będzie znacznik N, komórka $40000 zostanie ustawiona na $ff, w przeciwnym wypadku po wykonaniu tej instrukcji jej zawartość będzie równa 0.

■ **ABCD Dx,Dy „Add Decimal with Extend" - dodawanie dziesiętne z rozszerzeniem. -(Ax),-(Ay).**

Instrukcja ta dodaje w kodzie BCD bajt określony operandem źródłowym, do operandu przeznaczenia i tam też pozostawia wynik operacji. Dodawanie odbywa się z uwzględnieniem bitu rozszerzenia X.

Istnieją jedynie dwa tryby adresowania tej instrukcji:
1. zarówno operandem źródłowym, jak i operandem przeznaczenia są rejestry danych.
2. Operandy adresowane są przy pomocy trybu predekrement.

Czym jest kod BCD? Jest to sposób zapisu liczb dziesiętnych z wykorzystaniem kodu binarnego, w którym cyfra dziesiętna przedstawiona jest za pomocą czwórki bitów. System ten jest bardziej „rozwlekły" niż dwójkowy ze względu na to, że nie są wykorzystane kombinacje bitów w zakresie 10 - 15 (w systemie dziesiętnym nie istnieją cyfry o takich wartościach!!!). Ze względu na to, że kod BCD dla mikroprocesora nie jest naturalnym systemem liczenia, operacje arytmetyczne wykonywane w tym systemie zajmują więcej czasu. Większość procesorów posiada jednak wbudowane instrukcje operowania na tym kodzie, gdyż jest on bardzo rozpowszechniony we wszelkiego rodzaju urządzeniach elektronicznych: zegarki, układy zliczające, wyświetlacze cyfrowe.

A oto kilka przykładów zapisu liczb w kodzie BCD:

```
dec 34 = 00110100 BCD
dec 99 = 10011001 BCD
dec 10 = 00010000 BCD
dec 58 = 01011000 BCD
```

Znaczniki:
X - Ustawiany, gdy wystąpi przeniesienie.
N - Nie używany.
Z - Zerowany, gdy wynikiem była liczba różna od zera. W przeciwnym wypadku nie zmieniany.
V - Nie używany.
C - Ustawiany, gdy wystąpiło przeniesienie.

Przykład:

abcd d0,d2 - liczba zapisana w kodzie BCD, w wymiarze bajtu (dwie cyfry) zostanie dodana do liczby zapisanej w rejestrze d2 i tam też zostanie umieszczony wynik.

■ **SBCD Dx,Dy „Substract Decimal with Extend" - odejmowanie dziesiętne z - (Ax), - (Ay) rozszerzeniem.**

Instrukcja ta wykonuje odejmowanie w kodzie BCD, wraz z bajtem rozszerzenia X. Tryby adresowania są analogiczne do tych przedstawionych przy ABCD. Odejmowanie można wykonywać na operandzie wielkości bajtu, czyli dwóch cyfrach dziesiętnych.

Znaczniki:
X - Ustawiany w przypadku wygenerowania pożyczki.
N - Nie używany.
Z - Zerowany, gdy wynik jest różny od zera. W przeciwnym wypadku nie zmieniany.
V - Nie używany.
C - Ustawiany w przypadku wygenerowania pożyczki.

Przykład:

SBCD -(a0),-(a1) - rejestry adresowe zostaną zmniejszone o 1, a następnie bajt wskazywany przez a0 zostanie odjęty w arytmetyce BCD od bajtu wskazywanego przez a1.

Zastosowanie tego trybu adresowania, oraz specyficznego sposobu działania znacznika Z pozwala na organizowanie pętli, których zadaniem będzie wykonywanie obliczeń na liczbach przekraczających wielkość bajtu.

■ **NBCD <adres efektywny> „Negate Decimal with Extend" - negacja BCD z rozszerzeniem.**

Instrukcja ta powoduje negację bajtu określonego adresem efektywnym, w kodzie BCD.

Znaczniki:
X - Ustawiany w przypadku wygenerowania pożyczki.
N - Nie używany.
Z - Zerowany, gdy wynik jest różny od zera, w przeciwnym wypadku nie zmieniany.
V - Nie używany.
C - Ustawiany w przypadku wygenerowania pożyczki.

Przykład:
NBCD (a0) - bajt wskazywany przez rejestr a0 zostanie zanegowany w kodzie BCD.

■ **EXT Dx „Sign Extend" - rozszerzenie znaku.**

Instrukcja ta rozszerza bit świadczący o znaku liczby. W przypadku operacji na słowie bit 7 rejestru danych zostanie skopiowany do bitów od 8 do 15. Natomiast jeśli rozmiar określimy jako „długie słowo" bit 15 zostanie skopiowany do bitów 16-31. EXT bardzo często używana jest wraz z instrukcjami dzielenia, gdyż te ostatnie nie dają 32-bitowych wyników.

Znaczniki:
X - Nie zmieniany
N - Ustawiany zgodnie z rozszerzanym bitem.
Z - Ustawiany, gdy wynikiem operacji bylo zero.
V - Zerowany.
C - Zerowany.

Przykład:
move.l #$0000ffe2,d0
ext.l d0 - po wykonaniu tych dwóch rozkazów zawartość d0 będzie równa $ffffffe2.

Jak już wspomniałem na początku, przedstawię dzisiaj procedurę sortowania bąbelkowego. Metoda ta wykorzystywana jest stosunkowo często, głównie ze względu na jej prostotę. Naturalną konsekwencją tego jest dość długi czas wykonywania, jednak w przypadku niewielu liczb nie ma to istotnegoznaczenia. Czytelnikowi proponuję rozbudowanie tego krótkiego programu na sortowanie nie tylko samych liczb, ale również tytułów plików, dat utworzenia zbiorów itp. I jeszcze jedna uwaga: program działa w arytmetyce bez znaku. Oznacza to, że liczba $ff (zazwyczaj interpretowana jako -1) okaże się większa od np. $1c. W przypadku, gdy użytkownika nie będzie satysfakcjonowało takie rozwiązanie wystarczy zmienić rozkaz BCC na BGT.

A oto procedura:

```
*********************
*Sortowanie bąbelkowe *
*********************
N=8       ;Ilosc elementow tablicy do sortowania
          ;pomniejszona o 1
Sortowanie:
          lea       Table(pc),a0
Start:
          moveq     #0,d0
          moveq     #0,d2
DoMain:
          move.b    $01(a0,d0.l),d1
          cmp.b     (a0,d0.l),d1
          bcc       NieWymien
          move.b    (a0,d0.l),$01(a0,d0.l)
          move.b    d1,(a0,d0.l)
          moveq     #-1,d2      ;-1 oznacza, ze wszystkie
                                ;nie zostaly jeszcze
                                ;uporzadkowane

NieWymien:
          addq.l    #1,d0
          cmp.l     #N,d0
          bne       DoMain
          tst.l     d2
          bmi       Start
          moveq     #0,d0
          rts
Table:
          dc.b      17,3,6,8,1,9,7,2,5
```

Jak działa algorytm sortowania bąbelkowego? Otóż przeszukuje on tablic wartości, porównując ze sobą kolejne dwa bajty. W przpadku, gdy ułożone są w kolejności malejącej (pierwszy większy od drugiego) zamienia je miejscami, inaczej pobiera dwa następne. Takich przebiegów robi tyle, aż stwierdzi, że wszystkie bajty ułożone są poprawnie (tzn. każdy następny jest większy, lub równy od poprzedniego) i wraca z podprogramu.

Przesortujmy teraz następującą sekwencję bajtów $02,$08,$07,$01 według powyższego algorytmu. Mikroprocesor wykona następujące kroki:

L.p.	Bajty sortowane	Bajty aktualnie porównywane	Zamieni?
1	$02 $08 $07 $01	$02 i $08	nie
2	$02 $08 $07 $01	$08 i $07	tak
3	$02 $07 $08 $01	$08 i $01	tak
4	$02 $07 $01 $08	$02 i $07	nie
5	$02 $07 $01 $08	$07 i $01	tak
6	$02 $01 $07 $08	$07 i $08	nie
7	$02 $01 $07 $08	$02 i $01	tak
8	$01 $02 $07 $08	$02 i $07	nie
9	$01 $02 $07 $08	$07 i $08	nie

Proszę zauważyć ileż porównań było niezbędnych do przesortowania tylko czterech bajtów! - a przy tym ileż z nich zostało wykonanych wydawało by się może niepotrzebnie. Nie dziwmy się zatem, jeżeli do przesortowania kilku kilobajtów komputer potrzebować będzie kilkunastu minut. A może uda Wam się przyspieszyć tą procedurę?

„K.K." Kobuz

WŁASNE DEMO

W dzisiejszym odcinku tego cyklu chciałbym przyszłym „demopisarzom" udzielić kilku rad i wskazówek, które to - być może - sprawią, iż ich produkty staną się bardziej efektywne. Kilkakrotnie pisałem już na łamach „64+4", ale jeszcze raz powtórzę, że najważniejszą sprawą dla kodera jest oszczędność czasu. Cała filozofia napisania dema sprowadza się do tego, aby wszystkie procedury wykonywały się szybciej niż jedną ramkę. W tym celu koderzy uciekają się do wielu trick'ów, i im tylko znanych metod, aby przyspieszyć wektorówkę, sinus scrolla lub wektor bobsy. Pamiętajmy, że nie ma takiej procedury, której nie można by przyspieszyć.

Zastanówmy się teraz, jakie cechy powinno zawierać dobrze napisane demo.

1. Starajmy się jak najmniej używać rozkazów skoków bezwarunkowych. Obniżają one znacznie czytelność programów, dlatego w programie zawierającym wiele takich rozkazów, trudno jest wykryć nawet proste błędy. W przypadku, gdy ominięcie takiego rozkazu jest niemożliwe próbujmy JMP zastąpić BRA. Ten drugi jest krótszy i zarazem szybszy.
2. Rozkazy mnożenia i dzielenia zastępujmy dodawaniem, odejmowaniem, obrotami lub przesunięciami bitów, ewentualnie tablicujmy wyniki. Zajmie to stosunkowo dużo miejsca, ale wykonywane będzie ok. 8 razy szybciej!!! Instrukcje MULU, MULS, DIVU, DIVS wykonywane są przez mikroprocesor najwolniej.
3. W miarę możliwości rozbudujmy w swoim demie sieć podprogramów. Uzyskamy przez to dobrą czytelność oraz oszczędność pamięci. Piszmy je tak, aby po małych przeróbkach mogły współpracować z innymi naszymi demami. W ten sposób stworzymy bank własnych procedur i napisanie każdego następnego programu będzie kwestią pół godziny.
4. W swoich source'ach nie oszczędzajmy miejsca na komentarze. Na pewno po pewnym czasie zdarzy się, że zechcemy udoskonalić stare procedury. Jeżeli uprzednio nie zaopatrzyliście ich w znaczące etykiety i objaśnienia do poszczególnych części, zapewniam, iż sporo czasu upłynie zanim zorientujecie się „o co w tym wszystkim chodzi".
5. Przerwania używajmy tylko w ostateczności. Ich wygenerowanie, oraz dodatkowe procedury obsługi zabierają procesorowi cenny czas.
6. Pamiętajmy, że włączenie dodatkowych bit-plane'ów, sprite'ów lub innych kanałów DMA również pochłania dużo czasu. Warto czasem zrezygnować z trybu wysokiej rozdzielczości, aby przyspieszyć pracę procedur.
7. Wszystkie operacje i obliczenia wykonujmy nie korzystając z przechowywania wyników pośrednich w komórkach pamięci. O ile to możliwe wykorzystujmy za wszelką cenę rejestry procesora zarówno w przypadku operandów źródłowych, jak i przeznaczenia.

Krzysztof Kobus „K.K."

GRACZ DOSKONAŁY

Witam po raz kolejny w naszej rubryce dla graczy - leniwców, dziś znów nadgryziemy parę gier bez użycia ostrych narzędzi.

CONTINENTAL CIRCUIT

W czasie, gdy zapali się pierwsze czerwone światło pchnij joystick do przodu. Gdy zapali się drugie światło czerwone puść joysick. Po zapaleniu się zielonego pchnij joystick do przodu. Im szybciej zareagujesz, tym lepsze będziesz uzyskiwał przyspieszenie.

DATASTORM

Po załadowaniu poczekaj, aż ukaże się tabela wyników. Teraz wciśnij F 10 by otrzymać tajną wiadomość.

DENARIS

By włączyć wersję treningową wciśnij „Z", po wyborze gry. Następnie podłącz mysz do portu 2 i trzymaj wciśnięty prawy przycisk w czasie, gdy gra się ładuje.

DRIVING FORCE

Gdy pojawi się główne menu, użyj pointera i click'nij na dwóch literach „I" w napisie „Driving". Gdy rozpoczniesz grę samochód nie będzie zjeżdżał z drogi, ale będzie mógł być uderzony przez innych.

DOGS OF WAR

W czasie gry wpisz „TIMBO". Teraz naciskając F5 wyłączasz kolizję sprite'ów.

DRAGONS SCAPE

Wciśnij TAB, a następnie „2" by przenieść się do następnej strefy.

DRAGON SPIRIT

Zatrzymaj grę wciskając F9. Teraz wpisz „DRAGON HEAD" i wciśnij F10.

NARCO POLICE

Na małym monitorze w czasie gry wprowadź następujące hasła:

NOENEMIG	- brak przeciwników,
MUNICION	- odnawia zapas amunicji.

Dla lubiących szukać podam, że nie są to jedyne hasła. Jeśli dobrze poszukacie w programie to możecie znaleźć hasła do wyłączenia kamer i karabinów maszynowych, oraz kilka innych.

Mr. Ref

KURS JĘZYKA C (cz.2)

Poprzedni odcinek zawierał: typy zmiennych, formatowanie wydruków - funkcją **printf()**.*W tej części cyklu porozmawiamy o tablicach i przyjrzymy się kilku następnym instrukcjom języka C.*

Język C pozwala definiować tablice (a więc szereg danych zapisanych pod wspólną nazwą; dostęp do określonej danej następuje poprzez odwołanie do nazwy i podanie indeksu) według następujacego wzoru:

```
int Tablica[100]; /* To jest tablica stu elementow */
```

Definicja powyższa określa tablicę zmiennych typu int, o 100 elementach, zapisaną pod nazwą Tablica.

> **UWAGA!**
> Indeks, a więc wartość pomiędzy nawiasami [ ], w definicji określa ilość elementów. Jednakże elementy te są indeksowane od ZERA! W naszym przypadku pierwszy element tablicy to Tablica[0], ostatni - Tablica[99].

Tekst umieszczony pomiędzy znaczkami: /*...*/ jest nazywany komentarzem. Do dobrego zwyczaju należy umieszczanie komentarzy w kluczowych miejscach programu. Nie chodzi tu o komentarz przy każdej instrukcji. Ważne jest, by opisywał on deklarowane zmienne, by znajdował się na początku deklaracji funkcji i komentował jej działanie itp. O tym, że jest to konieczne, przekona się każdy, kto w dwa miesiące po zakończeniu pisania programu nie będzie w stanie dokonać w nim żadnej poprawki.

Rozważmy przykład:

```
main()
{
 int loop;        /* Zmienna uzywana w instrukcji for */
 int Tablica[10]; /* Tablica dziesieciu elementow */

 for ( loop=0; loop<10; loop++ )
  {
   Tablica[loop] = loop;
   printf ("\nElement Tablica[%d] = %d", loop,
         /Tablica[loop]);
  }
}
```

Program deklaruje dwie zmienne: loop oraz dziesięcioelementową tablicę Tablica[10]. Do dyspozycji mamy więc 1(loop) + 10(Tablica) = 11 danych. Każdemu elementowi tablicy zostaje nadana wartość równa indeksowi, tj. element Tablica[0] jest równy 0, element Tablica[1] jest równy 1, element Tablica[2] jest równy 2 itd.

Nadanie wartości poszczególnym elementom tablicy następuje wewnątrz instrukcji **for**. Funkcja ta powoduje wykonanie fragmentu programu określoną ilość razy. Jeżeli fragment, który ma być wykonywany, składa się z więcej niż jednej instrukcji, musi zostać ujęty w nawiasy { }, jeżeli składa się tylko z jednej instrukcji, nawiasy można pominąć.

Przykład:

```
main()
{
 int loop;        /* Zmienna uzywana w instrukcji for */
 int Tablica[10]; /* Tablica dziesieciu elementow */

 for ( loop=0; loop<10; loop++ )
   Tablica[loop] = loop;
}
```

Tu wewnątrz instrukcji **for** mamy tylko jedną instrukcję - nadania wartości (przypisania).

Format instrukcji **for** jest następujący:

```
for (PRZYPISANIE, WARUNEK, SKOK)
{
 instrukcje...
}
```

Tak więc instrukcje **for** z poprzednich przykładów można przeczytać jako: nadaj zmiennej loop wartość 0 (loop=0;), pętlę wykonuj dopóki zmienna loop jest mniejsza od 10 (loop;), a po każdym wykonaniu pętli zwiększ wartość zmiennej loop o jeden (loop++). Operator ++ jest operatorem inkrementacji i zwiększa wartość zmiennej o jeden.

> **UWAGA!**
> Zapis
> loop++;
> oznacza w tym wypadku: zwiększ o jeden wartość loop PO wykonaniu pętli.
> Zapis
> ++loop;
> oznaczałby: zwiększ o jeden wartość loop PRZED wykonaniem pętli. W tym przypadku program nie nadawał by wartości zmiennej Tablica[0].
> Operator **zmienna++** nazywany jest operatorem postinkrementacji, operator **++zmienna** jest operatorem preinkrementacji. Analogicznie do ++ działa operator --, jednakże efektem jego działania jest zmniejszenie wartości zmiennej o jeden.

Instrukcja **for** jest jednym z pożyteczniejszych narzędzi przy pisaniu programu. Warto zwrócić uwagę, że w języku C instrukcja ta posiada większe możliwości niż analogiczna np. w BASIC'u. Dlaczego? Otóż nigdzie nie jest powiedziane, że umieszczone w instrukcji **for**

parametry muszą dotyczyć jednej zmiennej! Oznacza to, że możemy tej instrukcji używać do przypisania wartości jakiejś zmiennej, jednakże do warunku zakończenia pętli można użyć innej zmiennej, a jako skok podać dowolne wyrażenie, np. wartość zwracaną przez jakąś inną funkcję itp.

Przykład:

```
for ( loop=0; error!=1; )
 {
 instrukcje...
 }
```

Tu nadano zmiennej loop wartość zero (np. wymaga tego jakaś funkcja czy warunek wewnątrz pętli), przy czym opuszczenie pętli **for** nastąpi dopiero wtedy, gdy zmienna error jest równa jeden (operator != znaczy "różny od"). Jest oczywiste, że wewnątrz pętli powinna znajdować się jakaś funkcja lub wyrażenie, które by modyfikowało zmienną error, w przeciwnym razie pętla byłaby wykonywana w nieskończoność. W powyższym przykładzie pominięty został skok zmiennej!

A oto inny, jeszcze ciekawszy sposób użycia **for**:

```
for (;;)
 {
 }
```

Pominięto tu WSZYSTKIE parametry pętli! Taka pętla jest po prostu pętlą nieskończoną! I często przydaje się w programach.

Przedstawimy teraz inne pętle dostępne w języku C - instrukcje **while** i **do...while**.

Format instrukcji **while** jest następujący:

```
 while ( warunek )
  {
   instrukcje...
  }
```

Oznacza ona polecenie: wykonuj instrukcje zawarte w pętli dopóki warunek jest prawdziwy. W tym sensie jest ona odpowiednikiem **for** (;warunek;). Wykonanie pętli odbywa się następująco: najpierw sprawdzany jest warunek, a potem, jeżeli jest on prawdziwy, następuje wykonanie pętli. Tak więc wnętrze instrukcji **while** w szczególnym przypadku może w ogóle nie być wykonane!

Zobaczmy, jak wygląda pierwszy przykład napisany przy użyciu **while**:

```
main()
{
 int loop;          /* Zmienna uzywana w instrukcji for */
 int Tablica[10];   /* Tablica dziesieciu elementow */

 loop=0;
 while ( loop < 10 )
  {
   loop++;          /* Zwiekszenie indeksu o jeden */
   Tablica[loop] = loop;
   printf ("\nElement Tablica[%d] = %d", loop,
         Tablica[loop]);
  }
}
```

Przed wykonaniem pętli **while** następuje nadanie wartości zmiennej indeksującej (loop=0;). Pętla jest wykonywana dopóki loop jest mniejsze od 10. Zwiększanie loop następuje wewnątrz pętli... Ale, ale!!! Od czego są operatory inkrementacji? Zwiększanie zmiennej loop możemy przecież umieścić w ... warunku!

```
main()
{
 int loop;          /* Zmienna uzywana w instrukcji for */
 int Tablica[10];   /* Tablica dziesieciu elementow */

 loop=0;
 while ( loop++ < 10 )
  {
   Tablica[loop] = loop;
   printf ("\nElement Tablica[%d] = %d", loop,
         Tablica [loop]);
  }
}
```

Teraz pętla **while** wykonywana jest następująco: sprawdź warunek, po jego sprawdzeniu i wykonaniu pętli, zwiększ wartość loop o jeden.

Tu uwaga o stylu. W miejscu, gdzie występuje operator inkrementacji lub dekrementacji warto go oddzielić spacją! Oczywiście nie należy tej spacji umieszczać pomiędzy operatorem a argumentem. Chodzi o to, by pisząc np.

```
c=a+++b;/* Fe, tak nie nalezy */
```

było wiadomo, o co chodzi, a więc czy jest to:

```
c = a++ + b;
```

lub

```
c = a + ++b;
```

Stąd też zapis (loop++ < 10), a nie (loop++<10).

Instrukcje **while** często wykorzystuje się do określania pętli nieskończonej:

```
while (1)
 {
  instrukcje...
 }
```

Podobna do **while** jest pętla **do... while** o formacie:

```
do
 {
 } while ( warunek );
```

Jest ona analogiczna do **while**, z jednym wyjątkiem. Otóż najpierw jest wykonywana zawartość pętli, a dopiero potem sprawdzany warunek. Oznacza to, że zawartość pętli **do...while** jest wykonywana zawsze minimum raz.

„W temacie" pętle to by było na tyle. Jeśli zajdzie potrzeba, dodatkowe informacje pojawią się przy kolejnych przykładach.

Wróćmy do tablic. Tablice w języku C są jednowymiarowe. Nic jednak nie stoi na przeszkodzie, by elementami tablicy były... tablice! Daje to możliwość definiowania tablic o dowolnych rozmiarach, np.

Tablica[10]	- Tablica jednowymiarowa o 10 elementach (0...9)
Tablica[10][10]	- Tablica dwuwymiarowa o 10*10=100 elemetach (0...9, 0...9)
Tablica[10][10][10]	- Tablica trzywymiarowa o 10*10*10 =1000 elementach (0...9, 0...9, 0...9)

Oto krótki program używający dwuwymiarowej tablicy. Zostanie ona zadeklarowana różnymi wartościami (0 lub 1), a następnie jej zawartość zostanie wyświetlona na ekranie:

```
/* Zaraz pojawi się szachownica */
main()
 {
   int x, y;              /* Dwie zmienne do indeksowania tablicy */
   char Szachownica[8][8];

   /* Nadajemy wartości poszczegolnym
   elementom tablicy */
   /* Pole czarne = 0; pole biale = 1 */

   for ( x=0; x<8; x = x + 2 )
    for ( y=0; y<8; y = y + 2 )
     {
     Szachownica[x]     [y]     =   0;
     Szachownica [x]    [y+1]   =   1;
     Szachownica [x+1]  [y]     =   1;
     Szachownica [x+1]  [y+1]   =   0;
     }
  /* Wydruk na ekran */
  for ( x=0; x<8; x++ )
   {
    printf ("\n");
    for ( y=0; y<8; y++ )
     if ( Szachownica[x][y] == 0 )
       printf ("0");
     else
       printf ("1");
   }
printf ("\n");
}
```

Pojawiła nam się w tym przykładzie instrukcja warunkowa **if...else.** Format **if** jest następujacy:

```
if ( warunek )
 {
  instrukcje1...
 }
else
 {
  instrukcje2...
 }
```

Oznacza ona: jeżeli **(if)** warunek jest spełniony, wykonaj instrukcję1, w przeciwnym razie **(else)** instrukcję2. Część instrukcji od słowa **else** włącznie jest opcjonalna i nie musi występować, np:

```
if ( i == 5 )
  printf ("\nWyobrazcie sobie, ze zmienna i jest rowna 5");
```

Konieczne jest odróżnienie OPERATORA PRZYPISANIA = (nadania zmiennej określonej wartości, np. x = 5) od OPERATORA PORÓWNANIA == (czy jest równe? np. x == 5 oznacza nie nadaj x wartość 5, lecz CZY X JEST RÓWNE 5). Naprawdę NALEŻY na to uważać, bo dostępne kompilatory na Amigę nie raczą nas ostrzec o możliwości wystąpienia takiego błędu.

Jarosław Chrostowski



W tym miesiący dotarły do nas kolejne prośby o pomoc w grach typu Adventure. Niektóre z odpowiedzi możecie znaleźć w starych numerach „64 plus 4". Poszukajcie!

Mam nadzieję, że weźmiecie udział w naszej zabawie w podpowiedzi. Tym razem nie publikujemy odpowiedzi bo ich nie znamy! Czekamy na listy od czytelników - na pomocną dłoń. Przypominam, że pomocy udzielamy zaznaczając kod prośby np. jeśli ktoś z Was wie co należy zrobić w grze Larry II to odczytuje kod (tu A2) pisze list z podpowiedzią i zaznacza „Odpowiedź na A2".

RATUNKUUUUU!!!

Police Quest I - jestem w biurze szefa Doodley'a i mam właśnie otrzymać nominację. Szef jednak nie może jej przeczytać bo na papierach usiadła kura. Co zrobić aby ją wypędzić i otrzymać nominację? Norbert Waligórski. **A1.**

Leisure Suit Larry II - wchodzę do basenu i topię się. Czy gdzieś jest może jakieś koło ratunkowe? Adam Żurkowski. **A2.**

Code Name „Iceman" - doszedłem do baru. Wiem, że muszę porozmawiać z brunetką. Jednak za każdym razem gdy się odzywam mąż blondynki wychodzi i „wali mnie w mordę". Jak to zrobić aby mówić do blondynki? Michał Stawny. **A3.**

Cruise For The Corpse - na zegarze mamy godzinę 9:10, zebrane dwa przedmioty (dwa świstki papieru), przeprowadzone kilka rozmów i .... Zrozpaczona i u kresu sił redakcja „64 plus 4". **A4.**

Sambor Kuźma

UWAGA! Dotyczy kasety AMIGA VIDEO SHOW.

Studio telewizyjne, które miało na nasze zlecenie przygotować kasetę AVS opóźniało jej przygotowanie. Kiedy przedstawiono gotową wersję okazało się, że jest tak niskiej jakości, iż nie nadaje się do powielania i dystrybucji. W związku z powyższym zwracamy wszystkim czytelnikom, którzy zamówili tę kasetę pieniądze. Prosimy równocześnie o wstrzymanie się od jej zamawiania. Wszystkich zainteresowanych serdecznie przepraszamy!

REDAKCJA

telegram

Wśród wszystkich, którzy zamówią komplet dysków

PUBLIC DOMAIN PACK

za rok 1991 na

AMIGĘ

- gwiazdkowa cena 240.000 zł

ROZLOSUJEMY

10 programów D-Mon professional

Do każdego zamówionego zestawu dołączamy gwiazdkowy prezent /gratis/ DISK BOX na 10 dyskietek 3.5"

Wśród wszystkich, którzy zamówią komplet dysków

PUBLIC DOMAIN PACK

za rok 1991 na

C - 64

- gwiazdkowa cena 200.000 zł

ROZLOSUJEMY

10 programów VOICETRACKER V4.0

Do każdego zamówionego zestawu dołączamy gwiazdkowy prezent /gratis/ DISK BOX na 10 dyskietek 5.25"

Wśród wszystkich, którzy zamówią komplet taśm

PUBLIC DOMAIN PACK

za rok 1991 na

C - 64

- gwiazdkowa cena 100.000 zł

ROZLOSUJEMY

10 programów VOICETRACKER V4.0

Wszyscy, którzy zakupią

D-Mon professional

w okresie gwiazdkowym otrzymują dodatkowo /gratis/

MOUSE PAD!

Nasza oferta gwiazdkowa ważna jest dla wpłat dokonanych od dnia ukazania się niniejszego numeru do 25 stycznia 1992 roku.

z ostatniej chwili

PUBLIC DOMAIN PACK

PUBLIC DOMAIN PACK C-64

Wrzesień

strona A:

- Mega Demo grupy FLASH

strona B:

- Hot Shot - magazyn dyskowy
- Code Sucker monitor - pr. użytkowy grupy PADUA
- Mountain Ride - gra w BASIC

Październik

strona A i B:

- MEGA DEMO „AIRDANCE 4" grupy T.A.T.

PUBLIC DOMAIN PACK TAPE NR 1

- SINUSDATA - EDITOR
- FAST CRUNCHER V3
- ANAL S.C. IBEYOND
- VECTOR - VICTORY
- PUZZLENOID +4
- TUNE OF MONTH #1
- NIM
- STRZAŁKA 64+
- LOGO - WRITER V.2.0
- CAN'T TOUCH IKU!
- INTRO PRV
- BONZIEED !!
- ZAX PACKIS
- READ THIS FIRST
- COMMERCIAL.BREAK
- 290 SPRITES!
- NOTE - ABOUT
- BAD NEWS NR2
- TO BAD NEWS...
- CONTACT CORNER!
- PROJEKT DUSZKÓW
- SYMPHONY NR 14
- SYMPHONY NR 15
- SYMPHONY NR 16
- SYMPHONY NR 17
- SYMPHONY NR 18
- SYMPHONY NR 19
- CRUISER/GIANTS
- NOTE > ANO < PADUA
- LET'S DYSP!
- FINALTAPE
- MUSIC - SEARCHER

PUBLIC DOMAIN PACK TAPE NR 2

- TURBO
- PUBL. DOMAIN. INFO
- FONTGRUB 1.0
- DREPTACZ BASIC
- LOAD DIS FIRSY
- MACROASSEMBLER
- TURBOASSEMBLER
- RELOCATOR
- LOGOPAINTER 3!
- REASSEBLER
- SPRITE - EDITOR
- FAST - CRUEL U.2.5
- HIGHLIFE NR5
- AXEL NEWS NR1
- GWIAZDY
- FLIGRAPH 2.2/BML
- NOTE TO FLI V.2.2
- DISKNOTKA/PADUA
- MEGA PACKER/T
- MIST II/ VISION
- TTECHSCR &DYSP
- PLASMA - WORLD
- VECTORBOBS...
- VECTOR - PLOTS
- FLI - UPSCROLL
- BORDER - HIRES
- ROCK AROUND
- FACEWRITER
- CHAR EDIT 2+2
- DISKNOTER
- DESTINATION'91
- CONTACTDEMO/ORE
- FONTEDITOR
- THE END

PUBLIC DOMAIN PACK AMIGA

Maj

- VIRUS X 5.0
- VIRUS TERMINATOR
- PARADOX - demo
- STORMCHILD - demo
- Moduły muzyczne:
  - MIAMI VOICE
  - ANTI ATARI SONG

Czerwiec

- POWER BOOT - własne menu dysku
- DISK CODING SYSTEM - program do zabezpieczania dysków
- Konwerter IFF - ANSI
- AUER NATION - demo
- Moduły muzyczne
- DOCS - opis gry ELWIRA
- LAMER DEFENCE - do wykrywania i niszczenia wirusów
- REWENG GO OF THE LAMER - grafika w trybie D_HAM

Lipiec

- Sanity - demo
- Amiga - Tanx (1Mb) - gra
- Little Beau (1MB) - gra
- There is A Light/Tonid - modules

Sierpień

- Real 3D - demo nowego programu do raytraceing'u
- Moduł Muzyczny XTC STEREO

Wrzesień

- MODUŁY MUZYCZNE dla programu TFMX:
  - > R - TYPE
  - > THE HOUSE OF TECHNO
- VIRUS EXPERT v181 + 143
- BOOT BLOCK'I
  - > BOOTX v 3.80
  - > IMPLODER v 4.0

Spis treści zestawów z poprzednich miesięcy - patrz wcześniejsze numery naszego pisma.
Zestawy „64 plus 4 PUBLIC DOMAIN PACK" można zamawiać wpłacając na konto: Bank PKO SA Oddział w Bydgoszczy konto nr:5.09011-400522.7-136-11-111.0 następujących kwot: 20.000zł za pojedyńczy zestaw dla C-64, 25.000zł za zestaw dla AMIGI. Kwoty te obejmują koszt dyskietki, koszty kopiowania, opakowania i przesyłki pocztowej. Blankiety wpłat powinny być **CZYTELNIE** wypełnione i zawierać: **imię i nazwisko, dokładny adres zamawiającego, skrót „PDP-64"**(jeśli zamawiamy zestaw dla **C-64**) lub **„PDP-A"**(zestaw dla Amigi) - dane te prosimy umieszczać na **wszystkich** odcinkach dowodu wpłaty. W prenumeracie zestawy kosztują: PDP-64 - 18.000zł (12 numerów 216tys zł), PDP-A - 22.000zł (12 numerów 264tys zł). Prenumeratę można zawrzeć w dowolnym terminie na okres od 3 do 12 miesięcy (do końca roku kalendarzowego). Prenumerata może obejmować miesiące od początku roku - tzn. zamawiając całoroczną prenumeratę np. w październiku, w pierwszej przesyłce otrzymacie wszystkie poprzednie zestawy.

ZAMÓW NIE ZWLEKAJ!